A Hybrid Memetic Approach for Fully Automated Multi-Objective Web Service Composition

Alexandre Sawczuk da Silva, Hui Ma, Yi Mei, Mengjie Zhang
School of Engineering and Computer Science,
Victoria University of Wellington
PO Box 600, Wellington, New Zealand
{*sawczualex, hui.ma, yi.mei, mengjie.zhang*}*@ecs.vuw.ac.nz*

***Abstract*—Service-oriented architecture (SOA) has been widely employed in the field of software engineering, since it encourages attributes such as modularity and code reusability across different applications. Web service composition, where atomic services that accomplish simple tasks are combined into an application that fulfils a more complex function, is one popular application of SOA. Existing composition works focus on building functional and quality-optimised applications. A subset of these works use multi-objective evolutionary computing techniques to produce a Pareto front of compositions, though they assume that the basic structure of the composition workflow is already known. In our recent work, we removed this assumption by combining NSGA-II with a solution representation that allows for different workflow structures to be investigated. However, the multi-objective nature of the problem made it difficult to apply local search to further improve the results. In this paper we hybridise NSGA-II with MOEA/D, which allows the problem to be decomposed into multiple single-objective subproblems where a simple form of local search can be applied. Experiment results show that the use of local search improves the quality of the fronts produced by the hybrid approach for a number of composition tasks.**

***Keywords*-NSGA-II; MOEA/D; Local search; Web service composition; QoS optimisation; Multi-objective; Combinatorial optimisation.**

I. Introduction

Service-Oriented Architecture (SOA) is a paradigm that has been widely adopted in recent years, as it encourages the creation of modular applications whose components can easily be reused [1]. The modules used in SOA are *Web services*, and they provide data or operations that can be independently accessed over the network [2]. The creation of modular applications within this paradigm is known as *Web service composition*, which entails the combination of services into a particular execution workflow [3]. In addition to creating workflows, the focus of the composition process is on creating solutions with the best possible overall Quality of Service (QoS). The need to efficiently optimise a solution's QoS makes evolutionary computing (EC) a promising composition approach, and several works have investigated this particular area [4].

The majority of existing EC approaches are classified either as single-objective or multi-objective works. Single-objective approaches generally prioritise the exploration of different composition workflow structures, meaning that prior knowledge on what structural features are most promising is not required. These approaches are referred to as *fully automated* Web service composition techniques [5]. Due to the varying structure and length of solutions, algorithms that support more flexible representations—such as genetic programming (GP) [6], [7]—are ideal for this task. Despite these advantages, one limitation of single-objective works is that they combine all potentially conflicting QoS measurements using a simple aggregation strategy, which assumes that the users who requested the composition have certain preferences regarding the importance of each QoS attribute. In fact, users could often benefit from being presented an array of possible choices. To address this need, another set of works has explored multi-objective techniques, since they can naturally handle conflicting QoS attributes and also produce a set of trade-off solutions [8]. These works assume an abstract workflow is given, and they simply select a service for each abstract slot until reaching the best possible overall QoS. These are known as *semi-automated* Web service composition techniques [5]. A recent work has employed a GP-based NSGA-II approach to composition [9], combining fully automated Web service composition with multi-objective principles.

Works in other multi-objective combinatorial optimisation problems [10] have shown that the use of local search further improves the quality of solutions, however this has not been attempted for multi-objective and fully automated Web service composition. Thus, the goal of this paper is to propose a memetic multi-objective Web service composition approach that employs local search to produce compositions with better QoS trade-offs. While a non-dominated form of local search can be directly applied to the entire population of NSGA-II, this has been shown in the past to be quite computationally expensive if not bound in some way [11]. Instead, this work employs a previously proposed NSGA-II and MOEA/D hybrid for the optimisation process [12]. The inclusion of MOEA/D is advantageous because it divides the multi-objective problem into single-objective subcomponents, enabling the use of less computationally expensive single-objective local search operator. The effectiveness of the memetic approach is verified by testing it against existing multi-objective works using the popular benchmark dataset WSC-2008 [13].

The remainder of this paper is organised as follows: Section II defines the problem addressed in this paper. Section III discusses relevant works in this area. Section IV presents the novel hybrid memetic approach. Section V describes the experimental design when comparing the novel approach with existing works. Section VI analyses the experimental results. Section VII concludes the paper.

II. Problem Description

Web service composition is centred around the idea of having a user send a *composition request* that is automatically processed by a system that then returns an application assembled using a set of atomic services. Given a composition request $R = \langle input(R), output(R)\rangle$, with $input(R)$ representing the inputs provided to the composition and $output(R)$ representing the outputs required from the composition. Each Web service used in the composition is represented as $S = (intput(S), output(S), \{q_1, ..., q_n\})$, meaning that it requires a set of *inputs*, produces a set of *outputs*, and has an associated set of *QoS* attributes describing how well it operates. The k services that are available for use are held in a *service repository* $SR = \{S_1, ...S_k\}$. The Web service composition process consists of selecting a subset of services from SR that provide the needed functionality, then combine them into a workflow that produces $output(R)$ when given $input(R)$.

A composition can be executed in its entirety when provided with $input(R)$, producing $output(R)$. This is referred to as *functional correctness*. Service compositions can be represented as directed acyclic graphs with a start service $S_s = (\emptyset, input(R), \emptyset)$ and an end service $S_e = (output(R), \emptyset, \emptyset)$. In order for a composition to be functionally correct, all other services must have their inputs provided by *predecessors*, which in turn have already had their inputs satisfied. Figure 1 shows an example of a functionally correct service workflow. The travel planning scenario shown here is popular in the field [14], [15], and the goal is to book flights and accommodation for a customer's trip according to their preferences.

In addition to functional correctness, another requirement for Web service compositions is to achieve the best possible overall QoS. The composition should be optimised by using *objective functions* $f_1, ..., f_n$ to improve each QoS attribute. In this work, four popularly considered attributes [4] have been considered: availability (A), the probability of a service immediately responding to a request; reliability (R), the probability that the response produced by the service is accurate; time (T), the overall execution time for responding a request; cost (C), the financial cost incurred in requesting the execution of a service. In the scenario of a Web service

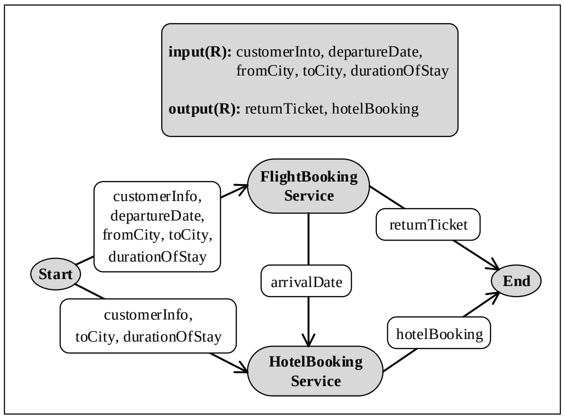
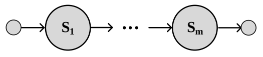

input(R): customerInto, departureDate, fromCity, toCity, durationOfStay
output(R): returnTicket, hotelBooking
Start
customerInfo, departureDate, fromCity, toCity, durationOfStay
FlightBooking Service
returnTicket
arrivalDate
End
customerInfo, toCity, durationOfStay
hotelBooking
HotelBooking Service

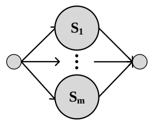
Figure 1: Web service composition example (adapted from [16]).

composition, the QoS values of its atomic services can be aggregated to calculate the overall QoS attributes for the composition. This is done according to the constructs that are used in the workflow, which are based on existing composition languages [17]. Two constructs are considered in this work.



$$A = \prod_{n=1}^{m} a_n \quad R = \prod_{n=1}^{m} r_n \quad T = \sum_{n=1}^{m} t_n \quad C = \sum_{n=1}^{m} c_n$$

Figure 2: Depiction of sequence construct and formulae for aggregating atomic QoS values [18].



$$C = \sum_{n=1}^{m} c_n \qquad A = \prod_{n=1}^{m} a_n \qquad R = \prod_{n=1}^{m} r_n$$

$$T = MAX\{t_n | n \in \{1, \ldots, m\}\}$$

Figure 3: Depiction of parallel construct and formulae for aggregating atomic QoS values [18].

Sequence Construct: The sequence construct connects services by using the outputs of a predecessor as the inputs of a successor, as seen in Figure 2. The A and R for this construct are obtained by multiplying together the corresponding atomic values for each service, and the T and C are obtained by adding up the corresponding atomic values.

Parallel Construct: The parallel construct groups services that are executed independently, as seen in Figure 3. This means that their inputs are fulfilled separately and their outputs have no bearing on each other. While A, R, and C are still obtained in the same way as they are for the sequence construct, T is calculated by finding the service in the construct with longest execution time (meaning that others will finish executing first).

The overall QoS values for a composition are used to calculate two objective values, based on previous works [9]. The two fitness functions are $f_1 = \bar{T} + \bar{C}$ and $f_2 = \bar{A} + \bar{R}$, both calculating scores between [0,2], where 0 is the best possible quality. The QoS attributes in both of these functions are normalised using a set of estimated bounds for each objective: the minimum A and minimum R are set to 0; the minimum T and minimum C are set to the lowest respective values found in a single relevant service in the repository; the maximum A and maximum R are set to the highest respective values found in a single relevant service in the repository; the maximum C and maximum T are also set to the highest single values found in a relevant service, but they are then multiplied by the total size of the repository [7]. Since both f_1 and f_2 are minimisation functions, A and R are offset during the normalisation process so that smaller values denote better quality.

III. Related Work

This brief discussion of relevant previous works is divided into four groups according to their common characteristics. The first group comprises semi-automated Web service composition works that use multi-objective optimisation techniques, an area that has been relatively well investigated. A dynamic genetic algorithm-based method is proposed to independently optimise the time and cost of compositions in [19]. This work assumes that an abstract workflow already exists and so it focuses on selecting the best services to fulfil, observing reputation and reliability constraints. In [8], an algorithm named E3-MOGA is employed to optimise the throughput, latency, and cost of service compositions, also assuming an abstract workflow. This approach produces solutions for multiple user levels, creating fronts with evenly distributed solutions. While these works have the ability to optimise different objectives independently, they assume that the overall workflow of the composition has already been provided. This is a limitation because the selection of a suitable composition workflow requires specialist knowledge.

The second group comprises fully automated Web service composition works that use single-objective optimisation techniques, an area that has further room for investigation despite a number of interesting contributions. Genetic programming (GP) is used to evolve workflow structures in [6]. A context-free grammar is used to enforce an initial coherent structure, then functional correctness is promoted by using a penalisation strategy in the fitness function. GP is also employed by [7] to generate functionally correct solutions, though in this case genetic operators preserve the correctness of solutions and the fitness function is only responsible for optimising the overall QoS values. The work in [20] clusters all potentially useful services into a workflow according to their functionality, then it employs Cuckoo search to select the most promising service from each cluster. While these works have the ability to identify promising workflow structures without the need for human intervention, they do not optimise conflicting objectives in an independent fashion. Instead, they aggregate different QoS measures as a single score. This is a limitation because it does not allow for different quality trade-offs to be thoroughly investigated.

The third group comprises fully automated Web service composition works using multi-objective optimisation algorithms, and up to this point it consists of a single work [9]. This work represents a composition as a set of fragmented service workflows, which must then be assembled into the corresponding service workflow. Service fragments are optimised using NSGA-II with specialised crossover and mutation operators. Despite effectively exploring a variety of composition structures in a multi-objective context, the fragmented representation does not naturally lend itself to the addition of local search, since listing all possible fragment variations for a given service is a complex and inefficient task.

The fourth group comprises works that employ decomposition and local search to improve the performance of multi-objective solutions to combinatorial optimisation problems. These works are not directly related to Web service composition, yet the principles they discuss are potentially quite useful in the composition domain. Perhaps one of the simplest ideas to boost the performance of a search carried out using NSGA-II would be to incorporate a simple Pareto local search operator, though this has been shown to be very computationally expensive if carried out to completion [11]. Alternatively, MOEA/D [21] could be applied for decomposing the multi-objective problem into several single objective subproblems, as this would enable a more efficient single-objective local search operator to be applied during the evolutionary process. Despite this advantage, the classic MOEA/D algorithm is known to be sensitive to the scale of each objective [22], which makes it potentially unsuitable to the Web service composition problem with its disparate QoS attributes. Given these limitations, hybridising NSGA-II with decomposition, as proposed in [12], could be a good compromise between non-dominated optimisation and a less computationally expensive form of local search. To the best of our knowledge this strategy has not been previously applied to the Web service composition problem, leaving an interesting knowledge gap to be explored.

IV. A Hybrid Memetic Composition Approach

The approach proposed in this work consists of employing the hybrid algorithm discussed in [12] with a linear representation that encodes services as a queue [23]. While it retains the simple non-dominated sorting strategy from NSGA-II, it also decomposes the problem using the principles of MOEA/D. This enables single-objective local search operators to be employed with a given probability, which is a less computationally expensive alternative to multi-objective local search approaches. The sequential representation facilitates the creation of suitable neighbours for the local search operator, since the position of services in the sequence can be swapped in order to create a suitable neighbour. A key aspect of this representation is that it is encoded, meaning that a decoding step is necessary in order to obtain the corresponding composition workflow for a given sequence. Figure 4 shows an example of this representation, along with its corresponding workflow.

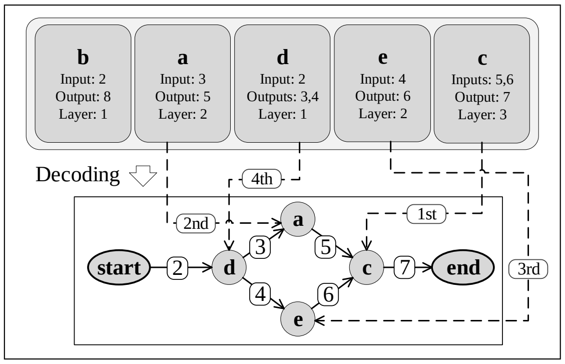


Figure 4: Example of candidate using sequential representation with corresponding composition workflow.

Figure 5 and Algorithm 1 show the overall steps of the proposed memetic approach. Initially the service repository preprocessed by dividing the services into layers, according to how close they are to the beginning node in the composition. This information obtained from this processing step is useful when decoding a sequence. Uniformly distributed weights are then generated for each subproblem, indicating the importance of each objective when evaluated in that context. Once a weight has been assigned to each subproblem, the Euclidean distance between all pairs of weights is calculated. Using this information, the closest T neighbours to each given weight are identified. Next, a population of N sequences is created at random. Since each sequence contains all the relevant services in the repository, a new candidate is created by simply shuffling the list of relevant services. Each newly created sequence is assigned to a different subproblem, and the QoS attributes of the newly created sequences are evaluated by decoding them. The following steps are then repeated until the stopping criteria, which in this work is reaching a specified number of generations, is met. Firstly, an empty set of individuals is created to hold the offspring resulting from the breeding process. Secondly, an offspring is created for each subproblem, has its QoS evaluated, and is placed in the offspring set. The creation of an offspring begins by selecting a parent via tournament selection. This is done by placing the subproblem sequence and its neighbours in a selection pool, then randomly picking some of these individuals for comparison. The individual that is evaluated as having the highest score for that subproblem is the tournament winner, and this is calculated by aggregating different the different objectives according to the weight vector using the Tchebycheff aggregation approach [21]. The winner has a genetic operator applied to it, with the specific operation being decided based on a predefined set of probabilities. The offspring resulting from this operation is then decoded, evaluated, and added to the offspring set. Thirdly, the offspring set is combined with the current population and the whole collection of individuals is sorted using the non-dominated strategy of NSGA-II, which includes the calculation of crowding distance. Fourthly, the N individuals with the best ranking after sorting are chosen to be the next generation. When the stopping criterion is eventually met, the non-dominated solutions in the final generation are returned as the final set of trade-off solutions.

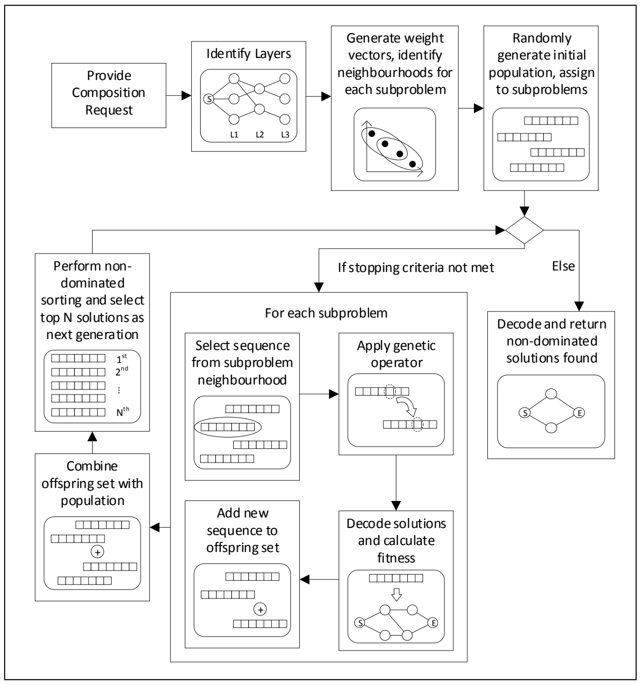

Provide Composition Request
Identify Layers
L1
L2
L3
Generate weight vectors, identify neighbourhoods for each subproblem
Randomly generate initial population, assign to subproblems
If stopping criteria not met
Else
Perform non-dominated sorting and select top N solutions as next generation
1st
2nd
Nth
For each subproblem
Select sequence from subproblem neighbourhood
Apply genetic operator
Decode and return non-dominated solutions found
Combine offspring set with population
Add new sequence to offspring set
Decode solutions and calculate fitness

Figure 5: Overall logical flow for the proposed hybrid memetic method.

A. Identification of Layers

As mentioned earlier, the service repository must be preprocessed before the evolutionary process takes place. The objective of this preprocessing step is twofold: to identify the services that are relevant to the composition task at hand, which determines the length of a sequence, and to identify the layer information associated with each service, which

Algorithm 1: Steps of the hybrid memetic composition algorithm.

```
Input  : R, SR, Population size $N$, neighbourhood size $T$
Output: Solution set
1: Separate the services in the repository into layers;
2: Generate $N$ uniformly distributed weight vectors, one for each subproblem;
3: For each weight vector, find its $T$ closest neighbours;
4: Randomly initialise a population of $N$ sequences;
5: Assign each sequence in the population to a subproblem;
6: Evaluate $f_1$, $f_2$ fitness of initial population by decoding each sequence;
7: while stopping criterion not met do
8:     Create an empty set of offspring;
9:     foreach subproblem do
10:        Place the subproblem sequence plus the neighbourhood sequences in a selection pool;
11:        Carry out tournament selection, evaluating candidates by aggregating objective values using subproblem weights;
12:        Apply a given genetic operator, randomly chosen according to probabilities, to the tournament winner;
13:        Evaluate result of genetic operation by decoding the sequence;
14:        Add result to set of offspring;
15:    Combine current generation with offspring set and perform the NSGA-II non-dominated sorting on the whole collection;
16:    Select top $N$ solutions as the next generation;
17: return non-dominated set of solutions in final generation;
```

is used during the sequence decoding process. The process for the identification of layers is discussed in detail in [23], and the overall idea is to classify services according to their distance to the composition's start node.

B. Sequence Decoding and Evaluation

A sequence of services is transformed into its corresponding composition workflow by using the decoding algorithm that is explained in detail in [23]. The key idea of this algorithm is to gradually build a workflow from the end towards the start, repeatedly going through the sequence. The sequence is searched and the services whose outputs can be used to fulfil the pending inputs in the workflow are added to the composition. The order of services in the sequence matters, as the search process always take place left to right (thus giving preference to the leftmost services). The overall QoS attributes for the composition are calculated during the decoding process, then returned and used to calculate the objective values as discussed in Section II.

C. Genetic Operators

Three problem-specific genetic operators were employed during the evolutionary process: mutation, crossover, and local search. The mutation operator randomly swaps two services in a sequence in order to create a sequence variation. The crossover operator used is a two-point crossover [23] that maintains the uniqueness of the elements in each sequence. The operation begins by randomly choosing a beginning and ending position for a subsequence to be exchanged between the parent sequences. The subsequences are copied into two empty offspring sequences (parent 1's subsequence is copied into offspring 1, and parent 2's into offspring 2). Subsequently, the offspring has its remaining blank cells filled from left right, reading from the complementary parent (also from left to right) and adding whatever services that have not already been included in the subsequence.

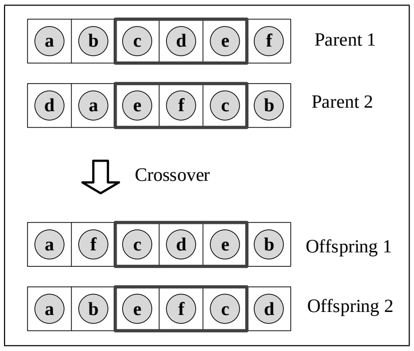


Figure 6: Crossover example for a pair of sequences.

The local search operator is based on the one described in [23], which consists of repeatedly swapping pairs of services as it is done in the mutation operator in order to create a neighbourhood. However, the local search proposed in this work will only swap services if two conditions are met: firstly, one of the services being swapped is used in the decoded composition and the other is not; secondly, the service that is not used in the composition has better QoS attributes overall than the service that has been included in the composition. The overall quality score for each particular service is calculated using the two objective functions introduced earlier, this time using the minimum and maximum of QoS values of individual services in the repository as normalisation bounds. The resulting values from these two functions are then combined using the Tchebycheff approach. This local search operator does not fix one of the services, thus the swap of all possible service pairs in the sequence is attempted with an optional maximum number of attempts. Figure 7 shows an example of this operator, where services a and b belong to the composition but service c does not. Each service has an associated quality score, and the smaller the score the better the quality. The local search attempts to swap all possible pairs of neighbours in the composition, which in this case amounts to three possible swaps. The first attempted swap (between a and b) is not successful because both services already belong to the composition, so neighbour 1 is not considered. The second attempted swap (between a and c) is also not successful, this time because service c's quality is worse than a's. The third swap (between b and c) is successful because one service belongs to the composition and the other does not, and c's quality is better than b's. Thus, the overall quality of neighbour 3 is assessed and, if better than that of the original candidate, the neighbour replaces it.

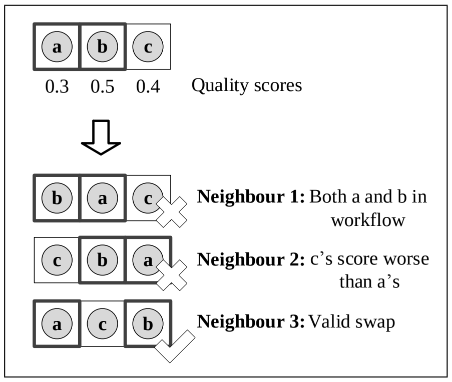

a
b
c
0.3 0.5 0.4
Quality scores
Neighbour 1: Both a and b in workflow
Neighbour 2: c's score worse than a's
Neighbour 3: Valid swap

Figure 7: Local search example for a sequence.

V. Experimental Design

Experiments were conducted to measure the performance of the proposed hybrid memetic approach (henceforth referred to as Hybrid-L), comparing it with NSGA-II using a sequential representation (henceforth referred to as NSGA-II) and with a version of the hybrid approach that does not include local search (henceforth referred to as Hybrid). The comparison is conducted the execution time of each approach, the IGD of the Pareto fronts—which is calculated using an aggregate front made up of the non-dominated solutions from all runs of all approaches for a given dataset—, and the hypervolume of the Pareto fronts—which is calculated using a reference point of (2,2). IGD and hypervolume were chosen because they are commonly used performance metrics for multi-objective optimisation [24]. The benchmark dataset WSC-2008 [13] was used for experiments, with the additional QoS attributes generated in QWS [7]. This dataset contains service repositories, taxonomies defining the relationships between inputs and outputs, and Web service composition tasks. The parameters used for executing the three approaches were selected based on commonly used settings in the evolutionary computing literature [25]. For all approaches, a population of size 500 was evolved for 51 generations and tournament selection was used with tournaments of size 2. For the NSGA-II and Hybrid approaches, the genetic operators used were mutation, crossover, and reproduction, with probabilities of 0.8, 0.1, and 0.1 respectively; for the Hybrid-L approach, the reproduction operator was replaced with the local search operator using the same probability. The local search stops after performing at most 100 iterations. For the Hybrid and Hybrid-L approaches, the neighbourhood was set to 500, and dynamic normalisation was used to calculate the fitness of the solutions, as discussed in [7]. Methods were run 30 independent times for each composition task on a personal computer with 8GB RAM and an Intel Core i7-4770 processor with a clock speed of 3.4GHz.

VI. Results and Discussions

The overall execution time in seconds for each approach is shown in Table I, which contains the mean and standard deviation for each composition task. Wilcoxon rank-sum testing with a significance level of 0.05 was used to identify which approaches produce the statistically best results for each composition task, and these results are bolded in the table. More specifically, a pairwise comparison of approaches without Bonferroni correction was carried, then approaches were ranked according to the number of times they were found to be better, similar, or worse than the others. The IGD results are displayed in a similar fashion in Table II, and the hypervolume results are shown in Table III. To clearly display the difference between the quality of the solutions produced by the different approaches, the objective values for all solutions were reset using QoS normalisation bounds retrieved from the set of all final compositions. Namely, the lower bounds for each QoS attribute are the lowest overall values found amongst all final compositions, and the upper bounds are the highest.

The execution time results show that the NSGA-II and Hybrid approaches are equivalent in performance for tasks 08-1, 08-2, 08-3, 08-8, 09-2, 09-3, 09-4, and 09-5, though for the remaining tasks the NSGA-II approach is significantly faster. The Hybrid-L approach, on the other hand, takes the longest to execute for all tasks due to the use of local search. In particular, tasks 08-6 to 08-8 and 09-3 to 09-5 are time consuming for Hybrid-L due to the high number of services in their repositories, which translate into longer sequences and higher numbers of possible neighbours during local search. Regarding the quality of the Pareto fronts produced, IGD results show that NSGA-II has significantly better values for 8 out of the 13 tasks and hypervolume results show that Hybrid-L has significantly better or equivalent values for 10 out of 13 tasks. Finally, in comparison with NSGA-II the Hybrid approach produces fronts that are equivalent in quality to each other overall, despite fluctuations within individual tasks.

Figures 8 and 9 depict the mean convergence of the IGD and hypervolume values for the NSGA-II, Hybrid,

Task (size)	NSGA-II	Hybrid	Hybrid-L
08-1 (158)	**0.64** ± **0.05**	**0.62** ± **0.05**	4.15 ± 0.58
08-2 (558)	**0.75** ± **0.1**	**0.73** ± **0.09**	4.52 ± 0.81
08-3 (604)	**3.63** ± **0.17**	**3.7** ± **0.17**	24.06 ± 1.31
08-4 (1041)	**1.09** ± **0.06**	1.28 ± 0.1	6.51 ± 0.33
08-5 (1090)	**5.08** ± **0.52**	5.68 ± 0.93	39.95 ± 4.33
08-6 (2198)	**16.3** ± **1.66**	18.59 ± 2.74	133.75 ± 15.87
08-7 (4113)	**16.62** ± **2.2**	18.61 ± 2.18	136.45 ± 20.78
08-8 (8119)	**56.2** ± **6.79**	**59.03** ± **6.96**	382.8 ± 75.39
09-1 (572)	**0.72** ± **0.06**	0.76 ± 0.06	4.53 ± 0.51
09-2 (4129)	**16.36** ± **2.6**	**16.92** ± **3.27**	111.88 ± 22.34
09-3 (8138)	**14.18** ± **1.79**	**15.2** ± **2.09**	202.29 ± 35.74
09-4 (8301)	**125.55** ± **13.79**	**128.8** ± **13.37**	835.39 ± 102.76
09-5 (15211)	**62.21** ± **10.42**	**64.34** ± **10.81**	407.73 ± 47.31

Table I: Mean and standard deviation for the execution time (s) of each approach.

Task (size)	NSGA-II	Hybrid	Hybrid-L
08-1 (158)	**0.05** ± **0.01**	0.16 ± 0.02	0.16 ± 0.01
08-2 (558)	**0.14** ± **0**	0.34 ± 0.2	0.35 ± 0.22
08-3 (604)	**0.03** ± **0.02**	0.06 ± 0.02	0.04 ± 0.02
08-4 (1041)	0.18 ± 0	0.02 ± 0.03	**0.01** ± **0.01**
08-5 (1090)	0.18 ± 0.11	0.1 ± 0.11	**0.04** ± **0.03**
08-6 (2198)	0.07 ± 0.03	0.08 ± 0.06	**0.06** ± **0.05**
08-7 (4113)	0.05 ± 0.02	**0.03** ± **0.02**	**0.03** ± **0.01**
08-8 (8119)	**0.07** ± **0.02**	0.17 ± 0.17	0.08 ± 0.11
09-1 (572)	0.04 ± 0.01	**0.01** ± **0.01**	**0.02** ± **0.02**
09-2 (4129)	**0.05** ± **0.07**	0.22 ± 0.06	0.2 ± 0.07
09-3 (8138)	**0.01** ± **0.01**	0.02 ± 0.01	0.01 ± 0
09-4 (8301)	**0.09** ± **0.02**	0.13 ± 0.03	0.11 ± 0.03
09-5 (15211)	**0.02** ± **0.01**	0.09 ± 0.02	0.09 ± 0.02

Table II: Mean and standard deviation for the IGD of each approach.

and Hybrid-L approaches when running task 08-5, in which Hybrid-L produced solutions with significantly better IGD and hypervolume values. In the original set of experiments the NSGA-II and Hybrid approaches execute significantly faster than the Hybrid-L approach. To verify whether the other approaches could achieve equivalent quality given longer execution times, NSGA-II and Hybrid were allowed rerun for 200 generations instead of 51, while the original settings were kept for Hybrid-L (all other parameters were unchanged). The plots included here were produced based on this experimental rerun. The IGD plot in Figure 8 shows that even given additional time, NSGA-II cannot reach the same IGD levels as the other approaches. The Hybrid approach, on the other hand, almost arrives at the IGD level of the Hybrid-L approach, but it eventually converges to a local optimum. The hypervolume in Figure 9 shows a similar pattern for NSGA-II, though this time the Hybrid approach converges faster than Hybrid-L and reaches an equivalent hypervolume to it.

VII. Conclusions

This paper introduced a hybrid memetic approach to QoS-aware Web service composition, combining elements of NSGA-II and MOEA/D to independently optimise quality attributes. The decomposition element of this approach

Task (size)	NSGA-II	Hybrid	Hybrid-L
08-1 (158)	**1.17** ± **0.01**	1.13 ± 0.03	1.14 ± 0.01
08-2 (558)	1.6 ± 0	1.59 ± 0.03	**1.62** ± **0.06**
08-3 (604)	**1.2** ± **0.06**	1.12 ± 0.06	1.18 ± 0.06
08-4 (1041)	1.24 ± 0	1.39 ± 0.01	**1.39** ± **0.01**
08-5 (1090)	1.19 ± 0.33	1.47 ± 0.32	**1.59** ± **0.05**
08-6 (2198)	1.02 ± 0.12	0.92 ± 0.24	**1.05** ± **0.21**
08-7 (4113)	1.35 ± 0.13	**1.42** ± **0.1**	**1.45** ± **0.01**
08-8 (8119)	0.78 ± 0.02	0.77 ± 0.17	**0.87** ± **0.11**
09-1 (572)	1.99 ± 0.02	**2** ± **0.01**	**2** ± **0**
09-2 (4129)	**1.36** ± **0.14**	1.06 ± 0.12	1.1 ± 0.15
09-3 (8138)	2.46 ± 0	2.48 ± 0.01	**2.48** ± **0**
09-4 (8301)	**0.25** ± **0.14**	**0.21** ± **0.16**	**0.3** ± **0.22**
09-5 (15211)	**1.21** ± **0.1**	**1.27** ± **0.01**	**1.28** ± **0.01**

Table III: Mean and standard deviation for the hypervolume of each approach.

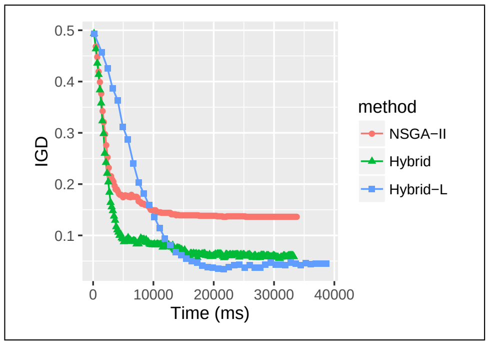


Figure 8: Mean IGD convergence over time for non-dominated solutions, for task 08-5. Smaller values indicate better quality.

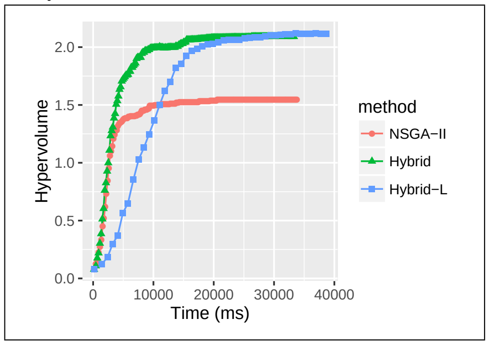


Figure 9: Mean hypervolume convergence over time for non-dominated solutions, for task 08-5. Larger values indicate better quality.

enables the employment of a new single-objective local search operator, which considers whether services are included in the composition as well as their overall quality. As shown by the experimental results, the quality of the solutions produced by the Hybrid-L approach (including local search) is improved for a number of composition tasks when compared to the NSGA-II and Hybrid (i.e. without local search) approaches. Even though the execution time required by the novel approach is higher than that of the others due to the local search operation, it still fits within a reasonable time budget. Thus, the ideal usage scenario for the Hybrid-L approach is when the focus is on the quality of the generated solutions. Future work in this area should investigate variations of the local search operator, aiming to reduce its required execution time, as well as exploring different neighbourhood sizes to be used by the tournament selection portion of the hybrid algorithm.

References

[1] L.-J. Zhang, J. Zhang, and H. Cai, "Service-oriented architecture," *Services Computing*, pp. 89–113, 2007.

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web services," in *Web Services*. Springer, 2004, pp. 123–149.

[3] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, 2007.

[4] L. Wang, J. Shen, and J. Yong, "A survey on bio-inspired algorithms for web service composition," in *Computer Supported Cooperative Work in Design (CSCWD), IEEE 16th International Conference on*. IEEE, 2012, pp. 569–574.

[5] J. Rao and X. Su, "A survey of automated web service composition methods," in *International Workshop on Semantic Web Services and Web Process Composition*. Springer, 2004, pp. 43–54.

[6] P. Rodriguez-Mier, M. Mucientes, M. Lama, and M. I. Couto, "Composition of web services through genetic programming," *Evolutionary Intelligence*, vol. 3, no. 3-4, pp. 171–186, 2010.

[7] H. Ma, A. Wang, and M. Zhang, "A hybrid approach using genetic programming and greedy search for QoS-aware web service composition," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XVIII*. Springer, 2015, pp. 180–205.

[8] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, "e^3: A multiobjective optimization framework for SLA-aware service composition," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 358–372, 2012.

[9] A. S. da Silva, Y. Mei, H. Ma, and M. Zhang, "Fragment-based genetic programming for fully automated multi-objective web service composition," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 353–360.

[10] G. Dhein, O. C. B. de Araújo, and G. Cardoso Jr, "Genetic local search algorithm for a new bi-objective arc routing problem with profit collection and dispersion of vehicles," *Expert Systems with Applications*, vol. 92, pp. 276–288, 2018.

[11] L. Ke, Q. Zhang, and R. Battiti, "A simple yet efficient multiobjective combinatorial optimization method using decompostion and pareto local search," *IEEE Trans on Cybernetics, accepted*, 2014.

[13] A. Bansal, M. B. Blake, S. Kona, S. Bleul, T. Weise, and M. C. Jaeger, "WSC-08: continuing the web services challenge," in *E-Commerce Technology and the 5th IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 10th IEEE Conference on*. IEEE, 2008, pp. 351–354.

[12] Y. Mei, K. Tang, and X. Yao, "Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 151–165, 2011.

[14] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven web services composition," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 411–421.

[15] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Transactions on software engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[16] A. S. da Silva, H. Ma, and M. Zhang, "GraphEvol: a graph evolution technique for web service composition," in *International Conference on Database and Expert Systems Applications*. Springer, 2015, pp. 134–142.

[17] P. Wohed, W. M. van der Aalst, M. Dumas, and A. H. Ter Hofstede, "Analysis of web services composition languages: The case of BPEL4WS," in *International Conference on Conceptual Modeling*. Springer, 2003, pp. 200–215.

[18] Y. Yao and H. Chen, "QoS-aware service composition using NSGA-II," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ACM, 2009, pp. 358–363.

[19] S. Liu, Y. Liu, N. Jing, G. Tang, and Y. Tang, "A dynamic web service selection strategy with QoS global optimization based on multi-objective genetic algorithm," in *International Conference on Grid and Cooperative Computing*. Springer, 2005, pp. 84–89.

[20] V. R. Chifu, C. B. Pop, I. Salomie, D. S. Suia, and A. N. Niculici, "Optimizing the semantic web service composition process using cuckoo search," in *Intelligent distributed computing V*. Springer, 2011, pp. 93–102.

[21] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on evolutionary computation*, vol. 11, no. 6, pp. 712–731, 2007.

[22] Q. Zhang, H. Li, D. Maringer, and E. Tsang, "Moea/d with nbi-style tchebycheff approach for portfolio management," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[23] A. S. da Silva, Y. Mei, H. Ma, and M. Zhang, "A memetic algorithm-based indirect approach to web service composition," in *Evolutionary Computation (CEC), IEEE Congress on*. IEEE, 2016, pp. 3385–3392.

[24] S. Jiang, Y.-S. Ong, J. Zhang, and L. Feng, "Consistencies and contradictions of performance metrics in multiobjective optimization," *IEEE transactions on cybernetics*, vol. 44, no. 12, pp. 2391–2404, 2014.

[25] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.